"Express Mail" mailing label number:

EL707907624US

SYSTEM FOR EMBEDDING PROGRAMMING LANGUAGE CONTENT IN VOICEXML

William J. Byrne, Mitsuru Oshima, Michael Achenbach, Beth A. Bottos and Dai Zhu

CROSS-REFERENCE TO RELATED APPLICATION

This application is related to and incorporates by reference herein in its entirety the commonly owned and concurrently filed patent application:

Attorney Docket Number M-9906 US entitled "OPEN ARCHITECTURE FOR A VOICE USER INTERFACE" by Mitsura Oshima.

BACKGROUND OF THE INVENTION

Interactive voice response services are an increasingly common part of everyday life. Services of this type are used to provide everything from banking and credit card information to interactive driving systems. Interactive voice response services are also an increasingly popular way to access the World Wide Web. This is true in spite of the growing popularity of personal data assistants and web enabled cellular telephones. The nearly ubiquitous availability of telephones, and the ability to use voice in non-traditional environments (such as when driving) ensures that the popularity and diversity of interactive voice response services will continue to grow.

Voice eXtensible Markup Language (VoiceXML) is a response to the increasing use of interactive voice response services. VoiceXML is a language for scripting interactive voice response services.

As an example, consider the following VoiceXML fragment:

```
<vxml>
    <form>
        <block>
            <prompt>
                Hello, World!
            </prompt>
        </block>
    </form>
</vxml>
```

When processed by a VoiceXML interpreter, the prompt portion of the script plays text-to-speech (TTS) "Hello, World!"

VoiceXML fragments of this type have proven to be a flexible mechanism for accomplishing many tasks. At the same time, there are important cases where VoiceXML is lacking in required flexibility. Consider, for example, the case of an interactive voice response email service. Designers of this type of service might wish to generate a prompt that welcomes each user by name and tells them how many emails they have received since their last visit (e.g., "Hello Mr. Smith, you have ten new emails."). Unfortunately, this type of prompt requires dynamic generation—it includes fields that must be changed to match each user and each number of new emails.

In fact, the non-dynamic nature of VoiceXML contributes to a range of implementation difficulties. These difficulties become more severe (in most cases) in more complex VoiceXML applications. As a result, there is a need for systems that include dynamic content in VoiceXML and similar languages. This need is particularly important for complex interactive voice response services.

SUMMARY OF THE INVENTION

An embodiment of the present invention provides a method for providing an interactive voice response service. The method uses a VoiceXML interpreter in cooperation with a voice/audio application. The voice/audio application uses scripts coded in VoiceXML with embedded Java Server Pages (JSP). The use of VoiceXML along with JSP allows the present invention to provide an implementation that minimizes interaction between the voice/audio application and the VoiceXML server.

For one embodiment, the for providing an interactive voice response service includes the steps of : creating a pool of audio prompts, subdividing the pool into segments, generating code to randomly select prompts from one of the segments, sending the generated code to a VoiceXML interpreter.

The foregoing has outlined rather broadly the objects, features, and technical advantages of the present invention so that the detailed description of the invention that follows may be better understood.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of a computer network shown as a representative environment for an embodiment of the present invention.

Figure 2 is a block diagram of a computer system as may be used in the network of Figure 1.

Figure 3 is a block diagram showing a set of software components deployed in the network of Figure 1.

The present invention may be better understood, and its numerous objects, features, and advantages made apparent to those skilled in the art by referencing the accompanying drawings. The use of the same reference symbols in different drawings indicates similar or identical items.

DETAILED DESCRIPTION

The preferred embodiments of the present invention and their advantages are best understood by referring to Figures 1 through 4 of the drawings. Like numerals are used for like and corresponding parts of the various drawings.

ENVIRONMENT

In Figure 1, a computer network 100 is shown as a representative environment for an embodiment of the present invention. Computer network 100 is intended to be representative of the complete spectrum of computer network types including Internet and Internet-like networks. Computer network 100 includes a number of computer systems, of which computer system 102a through 102f are representative. Computer systems 102 are intended to be representative of the wide range of large and small computer and computer-like devices that are used in computer networks of all types. Computer systems 102 are specifically intended to include non-traditional computing devices such as personal digital assistants and web-enabled cellular telephones.

Figure 2 shows a representative implementation for computer systems 102. Structurally, each computer system 102 includes a processor, or processors 200, and a memory 202. Processor 200 can be selected from a wide range of commercially available or custom types. An input device 204 and an output device 206 are connected to processor 200 and memory 202. Input device 204 and output device 206 represent all types of I/O devices such as disk drives, keyboards, modems, network adapters, printers and displays. Each computer system 102 may also includes a disk drive 210 of any suitable disk drive type (equivalently, disk drive 210 may be any non-volatile mass storage system such as "flash" memory).

Computer network 100 also includes a telephony subsystem 104. Telephony subsystem 104 may be implemented as a separate system (as shown in Figure 1) or, more typically, as a card included in one of computer systems 102. In either case, telephony subsystem 104 is connected to receive commands and data from computer system 102e. Telephony subsystem 104 is also connected to a telephone network 106. Telephone network 106 is connected, in turn to a telephone 108.

Computer network includes a series of software processes. As shown in Figure 3, these include media telephony speech (MTS) 302, VoiceXML interpreter 304 and one or more voice/audio applications 306.

MTS 302 is hosted by computer system 102e and interacts with telephony subsystem 104. MTS 302 provides one or more high-level interfaces to telephony subsystem 104, such as telephony, text-to-speech and speech recognition and prompt playing. Software processes in computer network 100 typically interact with telephony subsystem 104 through MTS 302.

VoiceXML interpreter 304 and voice/audio application 306 are hosted (together or separately) by one or more computer systems 102 in computer network 100. VoiceXML interpreter 304 sends requests to voice/audio application 306. Voice/audio application 306 sends VoiceXML responses to VoiceXML interpreter 304. VoiceXML interpreter 304 interprets the VoiceXML responses and forwards the interpreted results to MTS 302.

The combination of MTS 302, VoiceXML interpreter 304 and voice/audio application 306 provides an interactive voice interface for voice/audio application 306. Users access MTS 302 through telephone network 106 and telephony system 104. This access is bi-directional—users can both issue verbal/audio requests and receive verbal/audio responses. MTS passes user requests to VoiceXML interpreter 304 where they are translated into HTTP requests and passed to voice/audio application 306. Voice/audio application 306 responds to the HTTP requests with VoiceXML responses. Voice/audio application 306 sends these responses to VoiceXML interpreter 304 where they are translated into calls to MTS 302. MTS 302 passes these responses, in turn to telephony system 104 telephone network 106 and eventually, the user.

In general, it should be appreciated that the specific network of Figure 1, computer system of Figure 2 and software components of Figure 3 are all intended to describe a representative environment for a VoiceXML interpreter. Other, equally suitable environments may include these or other components. The specific functions of each component may also vary between different environments. Additional details

of the representative environment of Figures 1 through 3 are described in a copending US Application entitled "Open Architecture for a Voice User Interface." That disclosure is incorporate in this document by reference.

SYSTEM FOR EMBEDDING PROGRAMMING LANGUAGE CONTENT IN VOICEXML

Voice/audio application 306 uses a series of enhanced VoiceXML scripts. The enhanced VoiceXML scripts may include embedded elements of a high-level language. Voice/audio application 306 processes each script before sending it to VoiceXML interpreter 304 or in response to requests from VoiceXML interpreter 304. Voice/audio application 306 executes each high-level language (HLL) element as it is encountered during this processing.

As an example, consider the following VoiceXML fragment:

```
<vxml>
    <form>
        <block> You have email from <%=msg.getFrom()>%,
received at <%=msg.getDate()%>. Subject is
<%=msg.getSubject()%>
        </block>
    </form>
</vxml>
```

During processing, Voice/audio application 306 scans for the tokens "<%=" and "%>". Voice/audio application 306 treats text enclosed by these tokens as HLL expressions. Voice/audio application 306 evaluates (interprets) each HLL expression as it is encountered. Voice/audio application 306 replaces each HLL expression with its evaluated value.

After executing and replacing each HLL expression, the preceding fragment would have a form similar to:

```
<vxml>
    <form>
        <block> You have email from Joe Smith, received at 4 am.
Subject is airplane delay
        </block>
    </form>
</vxml>
```

After processing, voice/audio application 306 sends the resulting script to VoiceXML interpreter 304. VoiceXML interpreter then interprets the updated VoiceXML fragment.

The process of token recognition, HLL expression execution and HLL expression replacement can be adapted to a range of different languages. The examples used in this description focus on the use of the Java programming language

and the use of Java Server Pages (JSP) in particular. Java and JSP have the advantage of being widely adopted within the Internet programming environment.

The definition for Java Server Pages includes a range of tokens and directives. These include:

<%= Expression %>	Voice/audio application 306 evaluates expression and replaces it with the resulting value.
<% code %>	Voice/audio application 306 inserts code into the service method.
<%! Code %>	Voice/audio application 306 inserts code into the body of servlet class, outside of service method.
<%@ page att="val" %>	General setup directions for Voice/audio application 306.
<%@ include file="url" %>	Voice/audio application 306 includes the contents of the file identified by URL.
<%-- comment text --%>	Voice/audio application 306 ignores text between <%-- and --%>.
<jsp:useBean att=val />	Voice/audio application 306 finds or builds a Java Bean.
<jsp:setProperty att="val" />	Voice/audio application 306 sets bean properties.
<jsp:getProperty name="propertyName" value="val" />	Voice/audio application 306 finds and outputs bean properties.
<jsp:forward page="URL" />	Voice/audio application 306 forwards request to designated page.
<jsp:plugin attribute="val" />	Voice/audio application 306 generates code to

	request that an applet be run using a Java plugin.

Voice/audio application 306 can be configure to support all or a subset of these directives. The addition of Java Server Pages to VoiceXML provides a flexible framework for constructing voice applications. The efficiency of applications created with this framework can be greatly enhanced by creating VoiceXML scripts in a way that reduces interaction between VoiceXML interpreter 304 and voice/audio application 306. Each of the following examples has been selected with this consideration in mind. Code for the following examples is attached as Appendix A.

EXAMPLE ONE

```
<jsp:useBean id="random" class="com.genmagic.util.RandomPrompt">
    <jsp:setProperty name="random" property="addPrompt"
    value="hello_1.wav"/>
    <jsp:setProperty name="random" property="addPrompt"
    value="hello_2.wav"/>
    <jsp:setProperty name="random" property="addPrompt"
    value="hello_3.wav"/>
    .....
    <jsp:setProperty name="random" property="addPrompt"
    value="hello_29.wav"/>
    <jsp:setProperty name="random" property="addPrompt"
    value="hello_30.wav"
</jsp:useBean>
```

Execution of this fragment causes voice/audio application 306 to create a pool of audio files. In this case, thirty files are created but the same methods could be used with any number of files. This Java implementation for RandomPrompt subdivides the pool of files into segments. For this particular example, it may be assumed that six segments of five files are created.

VoiceXML scripts can generate code to select prompts from the prompt pool by including the expression:

```
<jsp:getProperty name="random" property="prompt"/>
```

During interpretation, Voice/audio application 306 replaces the getProperty directive with VoiceXML of the form:

```
<var name="tmp" expr="Math.random() * 4"/>
<if cond="1.0>=tmp">
    <audio src="builtin:hello_5.wav"/>
<elseif cond="2.0>=tmp"/>
    <audio src="builtin:hello_6.wav"/>
<elseif cond="3.0>=tmp"/>
    <audio src="builtin:hello_7.wav"/>
<elseif cond="4.0>=tmp"/>
    <audio src="builtin:hello_8.wav"/>
<elseif cond="5.0>=tmp"/>
    <audio src="builtin:hello_9.wav"/>
</if>
```

Execution of this code causes VoiceXML interpreter 304 to randomly select an audio prompt from a segment within the pool of audio prompts. This segment of code can be executed multiple times with the prompt chosen varying between the five prompts included in the segment. Each re-execution is accomplished without further interaction between VoiceXML interpreter 304 and voice/audio application 306.

EXAMPLE TWO

```
<jsp:useBean id="suffle" class="SufflePrompt">
    <jsp:setProperty name="suffle" property="addPrompt"
    value="hello_1.wav"/>
    <jsp:setProperty name="suffle" property="addPrompt"
    value="hello_2.wav"/>
    <jsp:setProperty name="suffle" property="addPrompt"
    value="hello_3.wav"/>
    .....
    <jsp:setProperty name="suffle" property="addPrompt"
    value="hello_29.wav"/>
    <jsp:setProperty name="suffle" property="addPrompt"
    value="hello_30.wav"/>
</jsp:useBean>
```

Execution of this fragment causes Voice/audio application 306 to create a pool of audio files. In this case, thirty files are created but the same methods could be used with any number of files. This Java implementation for SufflePrompt subdivides the pool of files into segments. For this particular example, it may be assumed that six segments of five files are created. Each segment includes a shuffled sequence of audio prompts. Thus, one segment might include audio prompts 29, 3, 15, 11 and 7. A second segment might include audio prompts 2, 24, 21, 7 and 10. Each segment is created without duplicate entries.

VoiceXML scripts can generate code to select prompts from the prompt pool by including the expression:

```
<jsp:getProperty name="suffle" property="prompt"/>
```

During interpretation, voice/audio application 306 replaces the getProperty directive with VoiceXML of the form:

```
<var name="tmp" expr="tmp + 1"/>
<if cond="tmp == 6">
    <assign name="tmp" expr="1"/>
</if>
<if cond="1 == tmp">
    <audio src="builtin:hello_29.wav"/>
<elseif cond="2 == tmp"/>
    <audio src="builtin:hello_3.wav"/>
<elseif cond="3 == tmp"/>
    <audio src="builtin:hello_15.wav"/>
<elseif cond="4 ==tmp"/>
    <audio src="builtin:hello_11.wav"/>
<elseif cond="5 ==tmp"/>
    <audio src="builtin:hello_7.wav"/>
</if>
```

The expanded code implicitly selects one of the segments from the pool of audio prompts. The first execution of this code by VoiceXML interpreter 304 causes the first prompt in the selected segment to be played. Subsequent executions cause the remaining prompts to be played in order. Each re-execution is accomplished without further interaction between VoiceXML interpreter 304 and voice/audio application 306.

EXAMPLE THREE

```
<field name="next">
<field>
    <% if (mail.isLast()) { %>
        <prompt> That was the last one </prompt>
    <% } else { %>
        <goto next="email.jsp#event=nextMail"/>
    <% } %>
</field>
```

This fragment is intended to be used to help an application (and user) navigate within a variable length list of audio prompts. Lists of this type are used in applications where the number of prompts is not fixed. An example might be a voice email application that uses different prompts for different items in a user's inbox. Since the number of inbox items changes, the list of prompts has no fixed length.

The VoiceXML fragment is configured so that VoiceXML interpreter 304 firsts tests to determine if the end of the list has been reached (i.e., there is no next item or no previous item). Only if this condition is false, does the VoiceXML interpreter 304 retrieve the next item from voice/audio application 306. This avoids interaction between VoiceXML interpreter 304 and voice/audio application 306 if there is no next/previous item.

EXAMPLE FOUR

This example applies to user interfaces that support multiple domains. User interfaces of this type are intended to allow users to perform multiple unrelated tasks, such as a reading email, listening to news and performing stock transactions. This type of application works best where the user is able to move randomly between domains (i.e., there is no preset order required).

In applications of this type, VoiceXML scripts do not know the identity of the next domain to be visited. This information can be generated by adding the expression:

```
<goto next="domain.jsp#event=nextDomain"/>
```

The following VoiceXML fragment can be used to generate different prompts for multiple domain user interfaces. The fragment causes VoiceXML interpreter to generate an initial prompt for a user's first visit to a domain. A secondary prompt is used for each subsequent visit.

```
<% if (domain.isCurrentDomainVisited() == false) { %>
<prompt> You have 3 new messages and old messages </prompt>
<% } else { %>
<prompt> You returned back to email. </prompt>
<% } %>
```

EXAMPLE FIVE

This example applies to systems that support multiple services (i.e., multiple voice/audio applications 306). Systems of this type work best if VoiceXML interpreter 304 performs initial routing for each incoming call. To perform this type of routing, a VoiceXML fragment of the following type may be used.

```
<% if (DNIS.startsWith("800123")) { %>
    <goto next="http://banking.genmagic.com/bankingService.jsp"/>
<% } else if (ANI.equals("4087744485")) { %>
    <goto next="http://my.genmagic.com/oshima.jsp"/>
<% } %>
```

This fragment causes VoiceXML interpreter to examine information about incoming calls. This information may include Direct Number Identification (DNIS), Automatic Number Identification (ANI) and User to User Identification (UUI) information. In each case, the information can be used to direct incoming calls to one or more voice/audio application 306.

```
<jsp:useBean id="router" class="com.genmagic.util.Router"/>
<jsp:setProperty name="router" property="dnis" param="dnis"/>
<jsp:setProperty name="router" property="ani" param="ani"/>
<jsp:setProperty name="router" property="uui" param="uui"/>


<% if (router.isServiceFor("onstar")) { %>
    <goto next="http://onstar.genmagic.com/login.jsp"/>
<% } else if (router.isServiceFor("bank")) { %>
    <goto next="http://banking.genmagic.com/bankingService.jsp"/>
<% } %>
```

CONCLUSION

While the invention has been described with respect to the embodiments and variations set forth above, these embodiments and variations are illustrative and the invention is not to be considered limited in scope to these embodiments and variations. Accordingly, various other embodiments and modifications and improvements not described herein may be within the spirit and scope of the present invention, as defined by the following claims.